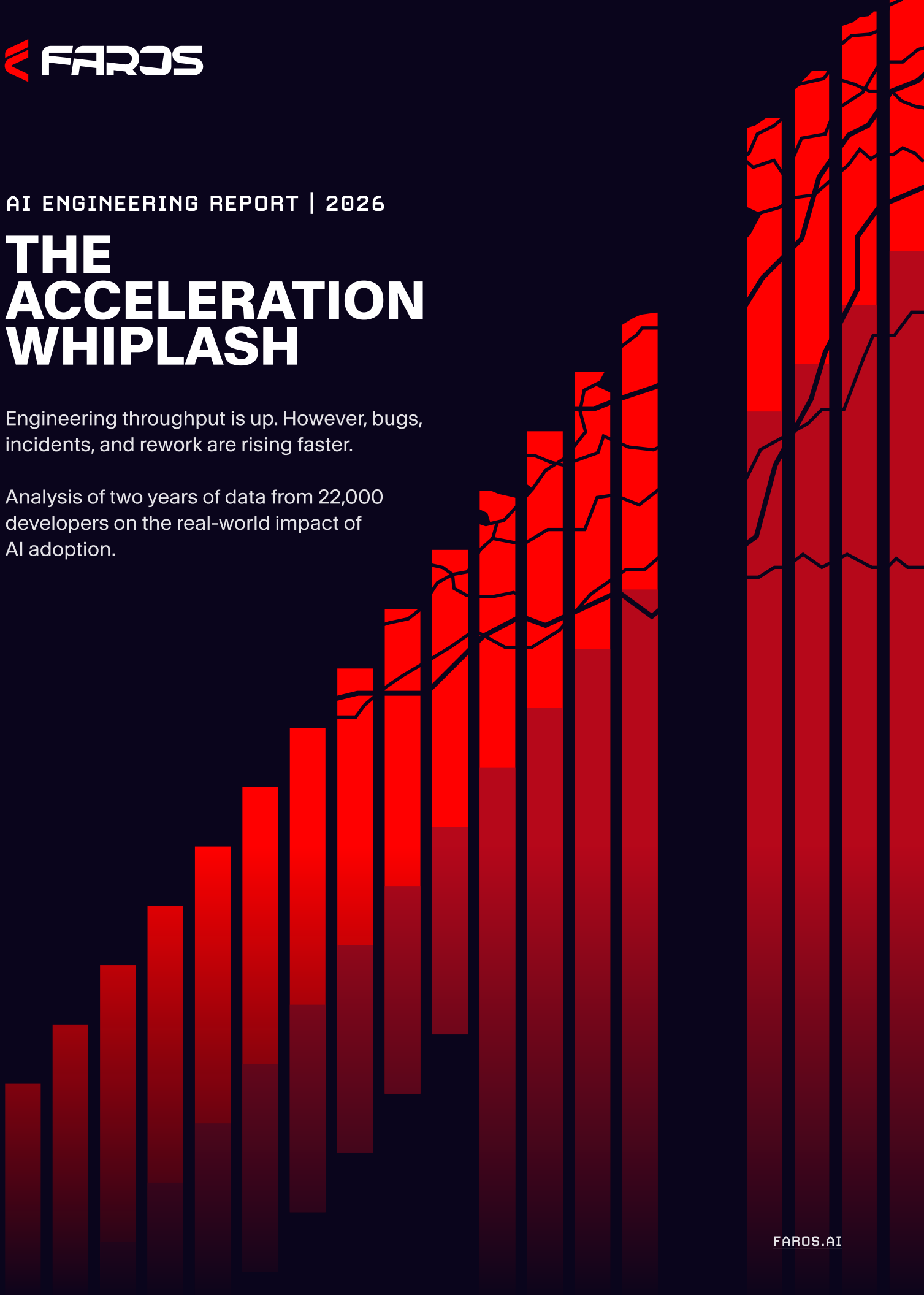
FAROS

AI ENGINEERING REPORT | 2026

THE ACCELERATION WHIPLASH

Engineering throughput is up. However, bugs, incidents, and rework are rising faster.

Analysis of two years of data from 22,000 developers on the real-world impact of AI adoption.



FAROS.AI

EXECUTIVE SUMMARY

More code. Not production-ready code.

When AI coding tools arrived, the promise was clear: developers would work faster, with an intelligent assistant at their side. The human was in charge, the AI suggested, and the human decided.

That is not what happened.

As model capabilities improved and vendor ambitions grew, the framing shifted. Co-pilots became peers. And without a deliberate decision by most organizations, AI crossed a threshold. With 60% of AI-generated code now being accepted into codebases, the distinction between assistant and author has collapsed in practice. Not an assistant. The author.

This report measures what that transition has produced, drawing on two years of data from 22,000 developers and 4,000 teams across the Faros platform.[1] The throughput numbers are undeniable. Task completion is up 34%. Epics completed per developer are up 66%. Tasks involving code specifically have increased 210%. AI is writing more code than ever.

What it is not doing is writing production-ready code.

Bugs per developer are up 54%. The incidents-to-PR ratio has more than tripled. Median review time has increased 5X. 31% more PRs are merging without any review. The relationship between AI adoption and production quality is worsening, not stabilizing, as adoption deepens.

We call this the Acceleration Whiplash. AI has flooded a system built around human-paced development and human-quality code with output it was never designed to absorb. The acceleration is real, but it is deceptive—it masks the strain building at every stage downstream. Adding reviewers, expanding QA, or investing in faster incident response treats the symptom. The problem must be addressed at the source, during code generation.

> **One finding holds across all segments of data:** All organizations suffer from the Acceleration Whiplash, irrespective of their engineering maturity. The organizations with strong pre-AI engineering practices see the same quality degradation as those without.

The question this report answers is not whether AI can write code. It can, at an extraordinary scale. The question is whether that code is good enough, and what that means for the decisions engineering leaders are making right now about headcount, process, and how far to extend AI's role in authorship.

[1] Organizations running on non-standard infrastructure were excluded.

Report's purpose

In July 2025, our first AI Engineering Impact Report identified what we called the AI Productivity Paradox: despite significant investment in AI tools, the expected productivity gains were not materializing in delivery outcomes. Our dataset at the time covered more than 10,000 developers and 1,255 teams.

This report returns to that question with a larger, more recent dataset of 22,000 developers across more than 4,000 teams we tracked over two years, and asks whether the pattern has changed. It has, but not in the direction most organizations are hoping for.

The paradox has sharpened into a crisis. AI adoption has accelerated. The gap between what AI produces and what engineering organizations can safely absorb has widened. And the numbers have gotten harder to ignore.

This report is for engineering leaders, CTOs, and the organizations advising them. We let the data make the argument, while keeping editorial commentary anchored to the numbers.

> **Note on continuity:** This study and our 2025 report are independent cross-sections of the industry, not a longitudinal panel. The datasets differ in size, time window, and composition. Where we reference comparisons to the prior report, we do so to indicate directional consistency, not to assert precise year-over-year trends.

Research methodology

This study analyzes the relationship between AI adoption and engineering outcomes across a large sample of development teams, drawing on approximately two years of telemetry data spanning the period of significant AI tool adoption across the industry. Telemetry was aggregated from task management systems, IDEs, static code analysis tools, CI/CD pipelines, version control systems, incident management systems, and HR system metadata.

The analysis focuses on development teams as they increased AI tool adoption above the 50% weekly active user threshold. AI adoption includes developer-facing coding assistants, including Claude Code, Cursor, GitHub Copilot, Windsurf, and similar tools, as well as autonomous agents integrated directly into the SDLC.

How we isolated the signal

- Standardized all metrics per company to remove inter-organizational variance.
- Used Spearman's rank correlation (ρ) to assess relationships between metrics and AI usage.
- Reported only metrics with data from at least 6 companies and statistically significant correlations (p-value < 0.05).
- For each team, we calculated the percent change in metric values between the two quarters of lowest AI adoption and the two quarters of highest AI adoption within the observation period.
- Excluded outlier data and metrics with insufficient historical coverage.

This approach enables comparisons within each company over time and avoids misleading aggregate assumptions across different organizational structures.

> **Versioning note:** This version of the report reflects analysis as of March 2026.

FINDINGS

More code. Declining quality. Accelerating incidents.

AI has not just accelerated software development. It has restructured who, or what, is doing the work. In the organizations we studied, AI has moved from a tool that assists engineers to one that authors code at scale, with humans increasingly in a review and oversight role rather than a creation role. That shift is visible in the throughput numbers. It is also visible in everything that follows.

The data in this report covers every stage of the software development workflow: how work is initiated, how code is written, how it moves through review and testing, and what happens when it reaches production. Across every downstream stage, the signal is the same: volume is up, quality is down, and the gap between the two is widening as adoption deepens.

A direct counterpoint to DORA's 2025 findings

DORA's 2025 State of AI-Assisted Software Development report concludes that AI amplifies existing strengths and weaknesses, and that strong engineering foundations offer protection against AI's downsides.[2] Our telemetry data, drawn from engineering systems across thousands of teams, does not support that as a protective factor. High-performing engineering organizations are experiencing the same downstream deterioration as everyone else.

The discrepancy is worth examining. DORA's findings are based on surveys. Surveys capture how developers feel about their work. And right now, developers feel more productive, because at the individual level, they are. Task completion is up. Code is flowing faster. The tools feel powerful.

What surveys cannot capture is what happens downstream: the review queues quietly backing up, the incidents accumulating in production, the bugs reaching customers that should have been caught earlier. Consider one finding from this report: 31.3% more PRs are being merged without any review. A gate only works if the gatekeeper is not buried under an avalanche. By the time those consequences show up in how people feel, months have passed and the signal is already stale.

This is the fundamental limitation of survey-based research during a period of rapid AI transformation. Perception lags reality, while telemetry does not. Engineering organizations making consequential decisions about headcount, tooling, and process need data that is as close to real time as possible, drawn from the systems where the work actually happens, not from how people feel about the work after the fact.

[2] DORA's 2025 State of AI-Assisted Software Development report

The maturity finding is the most striking. Organizations with strong pre-AI engineering performance, mature DevOps practices, high DORA scores, and disciplined delivery processes are not insulated from these effects. The whiplash appears regardless of baseline engineering maturity. Even the strongest foundations are buckling under the landslide of AI-generated output.

Our findings are organized in the following order:

1. AI adoption
2. Developer cognitive load
3. Throughput
4. Code complexity
5. Pre-merge code quality
6. Flow and efficiency
7. Code quality in production

`FINDING #1: ADOPTION`

AI is now the standard, not the experiment

80% of teams use AI regularly. Acceptance rate tripled. AI usage is now the norm.

AI tool adoption has accelerated sharply, and developers are accepting more of what AI produces than ever before.

State of AI adoption in software development

- **60%** of developers use at least one AI tool weekly (up from 40% in prior dataset)
- **80%** of teams exceed the 50% weekly active user threshold (up from 50% in prior dataset)
- **60%** acceptance rate of AI-generated code (up from 20% in prior dataset)
- **25%** of PRs are reviewed by AI agents (up from zero in prior dataset)
- **<1%** of PRs are opened by AI agents

The mechanism behind this shift is instructive. The percentage of developers holding a license to at least one AI tool has remained flat. Organizations had already bought the seats, so what changed is utilization: companies spent the past year actively driving adoption among developers who had access but were not yet using tools consistently.

> **Companies spent the past year actively driving adoption among developers who had access but were not yet using tools consistently.**

Code acceptance rates reflect the same acceleration. The overall rate of AI-generated code being accepted into codebases has risen from 20% to 60%. This increase is driven substantially by tools like Cursor and Claude Code operating in agent mode, where the agent applies changes directly.

> **The overall rate of AI-generated code being accepted into codebases has risen from 20% to 60%.**

Less than 1% of PRs are opened by AI agents, referring to when the AI agent independently writes, commits, and submits the code changes for review without human initiation. Agentic PR review has moved from zero to 25% of PRs, deployed significantly faster.

FINDING #2: DEVELOPER COGNITIVE LOAD

Thrashing and cognitive load are increasing

AI made it easy to start work. It did not make it easy to finish it.

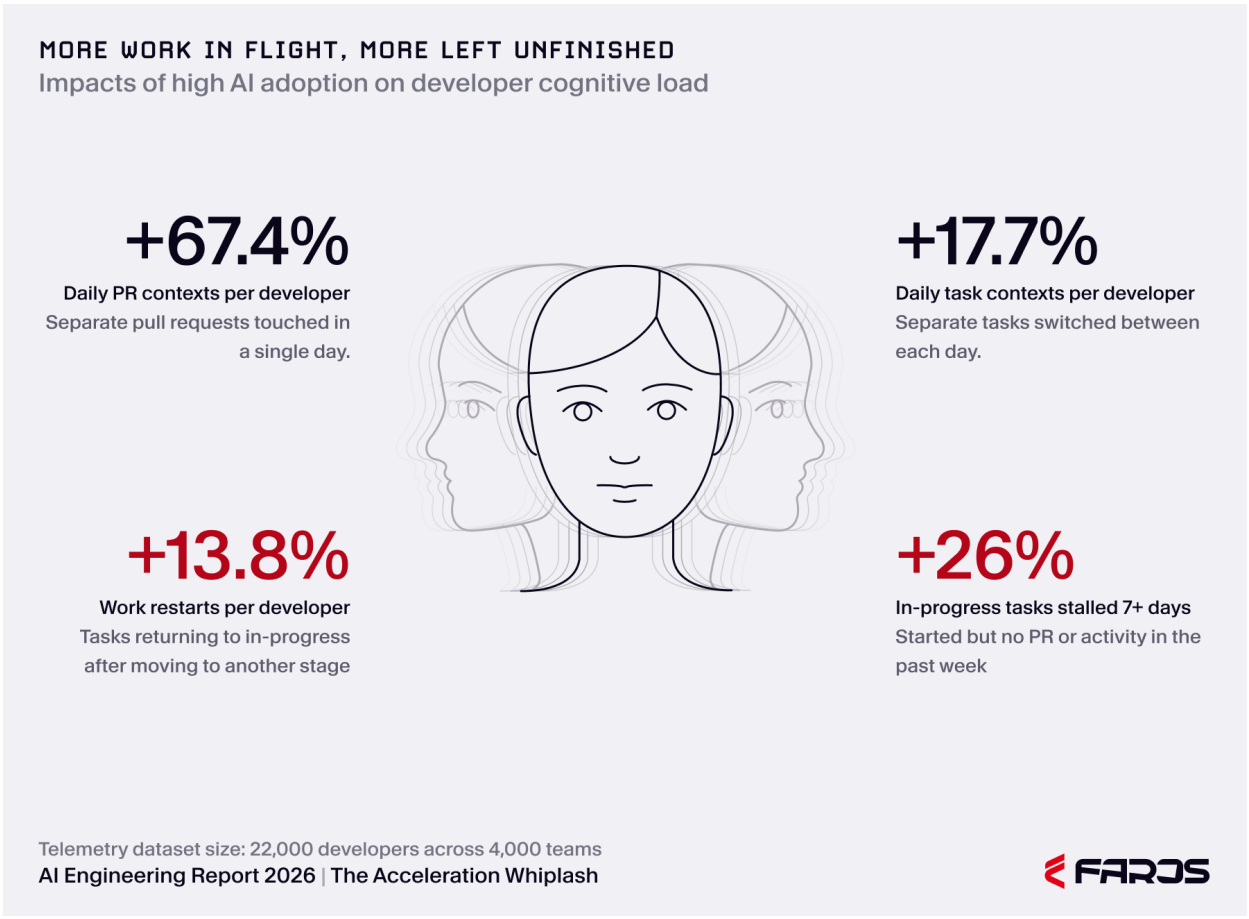
The research on context switching in software development is unambiguous: nearly every interruption forces developers to rebuild context before they can write a single line of code.[3]

The data shows that AI has added to this burden rather than reduced it. A well-functioning system should offload cognitive work from engineers, freeing them to focus on the decisions that require human judgment. Instead, what our data shows is engineers managing more concurrent threads, more parallel PRs, and more thrashing between states than at any prior point in our data.

Context-switching: metric % change as teams move from low to high AI adoption

+67.4%	daily PR contexts per developer (up from +46% in prior dataset)
+17.7%	daily task contexts per developer (up from +9% in prior dataset)
+13.8%	work restarts per developer (tasks returning to in-progress after another stage)
+26%	in-progress tasks with no PR or activity in the past 7 days

[3] Parnin, C., & Rugaber, S. (2011). Resumption strategies for interrupted programming tasks. Software Quality Journal, 19(1), 5–34. https://doi.org/10.1007/s11219-010-9104-9



Context switching was already rising in our 2025 report and has continued to accelerate. With high AI adoption, the number of PRs a developer touches per day has now risen 67.4%, and the number of tasks touched daily increased 17.7%. As developers become more capable of initiating work quickly with AI assistance, they tend to open more threads simultaneously. This is also a preview of what agentic workflows will intensify. Developers orchestrating multiple AI agents in parallel, reviewing their outputs, unblocking them, and deciding what to accept represents a natural evolution of the context-switching dynamic already visible in this data.

The consequences of increased cognitive load show up in the data. Work restarts are up 13.8%. These are tasks moving back to in-progress from another state. Restarts are expensive because the developer has to reload context they had already set down—reconstructing where they were, what they were trying to solve, and why they made the decisions they did. Meanwhile, 26% more in-progress tasks show no PR or activity for seven or more days: work that was started, claimed capacity, and then stalled. The picture is of a development environment where it is easy to begin and hard to finish.

26% more in-progress tasks show no PR or activity for seven or more days: work that was started, claimed capacity, and then stalled. The picture is of a development environment where it is easy to begin and hard to finish.

Despite this, throughput is up as the next section shows. AI's ability to accelerate individual output is real, but the thrashing visible in these numbers is a tax on that acceleration, and it is growing.

FINDING #3: THROUGHPUT

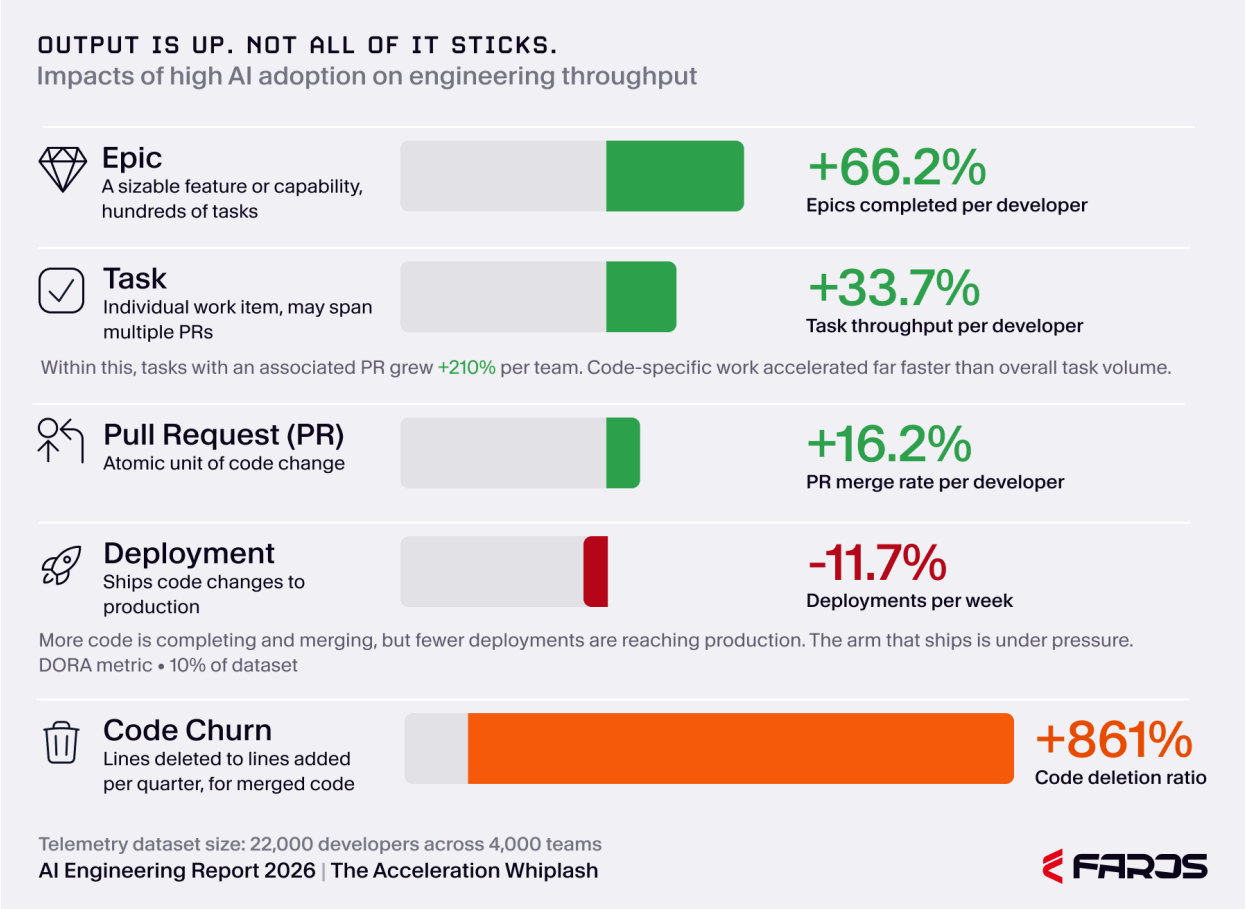
Output is up. Not all of it sticks.

Developers are doing more. Execution is accelerating. Code churn raises a question about what counts.

AI is meaningfully accelerating output. Task throughput, epic completion, and PR merge rate are all up. Deployment frequency is the exception. But one figure introduces a question the throughput numbers alone cannot answer: code churn, the ratio of lines deleted to lines added in a given quarter, has increased 861% with high AI adoption. The throughput numbers measure what was shipped. They do not tell you what survived.

Throughput: metric % change as teams move from low to high AI adoption

+33.7%	task throughput per developer
+66.2%	epics completed per developer
+16.2%	PR merge rate per developer
+210%	tasks completed per team with an associated PR
-11%	deployments per week (measured by 10% of the dataset)
+861%	code churn (lines deleted to lines added per quarter, for merged code)



Tasks of all types, including documentation, research, and design work, saw throughput rise 33.7% per developer. But tasks that involved code specifically, those with an associated PR, rose more than six times as fast at the team level. This is what AI tools are designed to do: accelerate the act of writing software. The gap between general task completion and code-specific task completion is itself a signal of where AI is having its most concentrated effect.

The improved epic completion rate is the single finding most directly tied to business outcomes. Epics represent large features, multi-sprint efforts that correspond to meaningful product capabilities. A 66.2% increase in epics completed per developer suggests AI is enabling teams to complete larger bodies of work, not just individual tasks.

> **A 66.2% increase in epics completed per developer suggests AI is enabling teams to complete larger bodies of work, not just individual tasks.**

One figure warrants context. PR merge rate per developer is up 16.2% after high AI adoption, but considerably lower than the 98% increase in our 2025 report. The most likely explanation is not diminishing returns from the tools, rather that the code quality is creating a review bottleneck that is throttling throughput. Organizations are generating more code than the system can review and merge.

For the subset of organizations that instrument deployment frequency (approximately 10% of the dataset), deployments per week are down 11.7%. More code is being written and merged, but it is not being released to production as frequently.[4]

Code churn, defined as the ratio of lines deleted to lines added for merged code in a given quarter, has increased 861% under high AI adoption. At 9.6 times the prior rate, significantly more code is being removed relative to what is being added.

There are several possible explanations for what is driving this increase, and the metric as measured in cross-customer observational research cannot resolve the ambiguity.[5] One explanation is that developers are accepting AI-generated code quickly, then returning to replace it when it proves insufficient in practice—rework that happens within the same measurement window and represents real waste. A second, more optimistic explanation is that AI is enabling teams to tackle large-scale refactoring projects that were previously too costly or too slow to staff. Legacy systems that had accumulated for years are finally being replaced, generating high deletion volumes that reflect productive architectural work rather than failure. A third possibility is that AI is simply accelerating the pace at which engineers return to improve code they were never fully satisfied with at the time of shipping. All three explanations are consistent with the data.

What organizations can do is investigate this in their own environment. With Git-level access to line provenance data, you can determine whether the deleted lines were written recently, suggesting rework of AI-generated code, or whether they represent legacy code being productively refactored. Analyzing the deletion-to-addition ratio per repo or application at monthly rather than quarterly intervals produces a stronger signal by narrowing the window in which the deleted code could have originated. Either way, a significant increase in this ratio is worth understanding. Throughput measures what was shipped, not what survived. Code churn is the asterisk on every output number in this section.

---

[4] Deployment frequency is measured by approximately 10% of the teams in our dataset. The findings are statistically significant within this subset and directionally consistent with the broader data. We include them because delivery-layer visibility will become increasingly important as AI adoption scales.

[5] This finding reflects cross-customer observational research conducted at the metadata level. Aggregate metrics across organizations provide directional signals without requiring deep access to individual codebases. Determining whether deleted lines were written recently or represent legacy refactoring requires Git-level line provenance data. This analysis is available to individual organizations conducting a deeper investigation within their own environment using Faros.

FINDING #4: CODE COMPLEXITY

Code changes are getting larger and more complex

Every structural dimension has grown. Simultaneously.

The code being produced is not only more voluminous. It is structurally more complex across every dimension we measure when AI adoption is high.

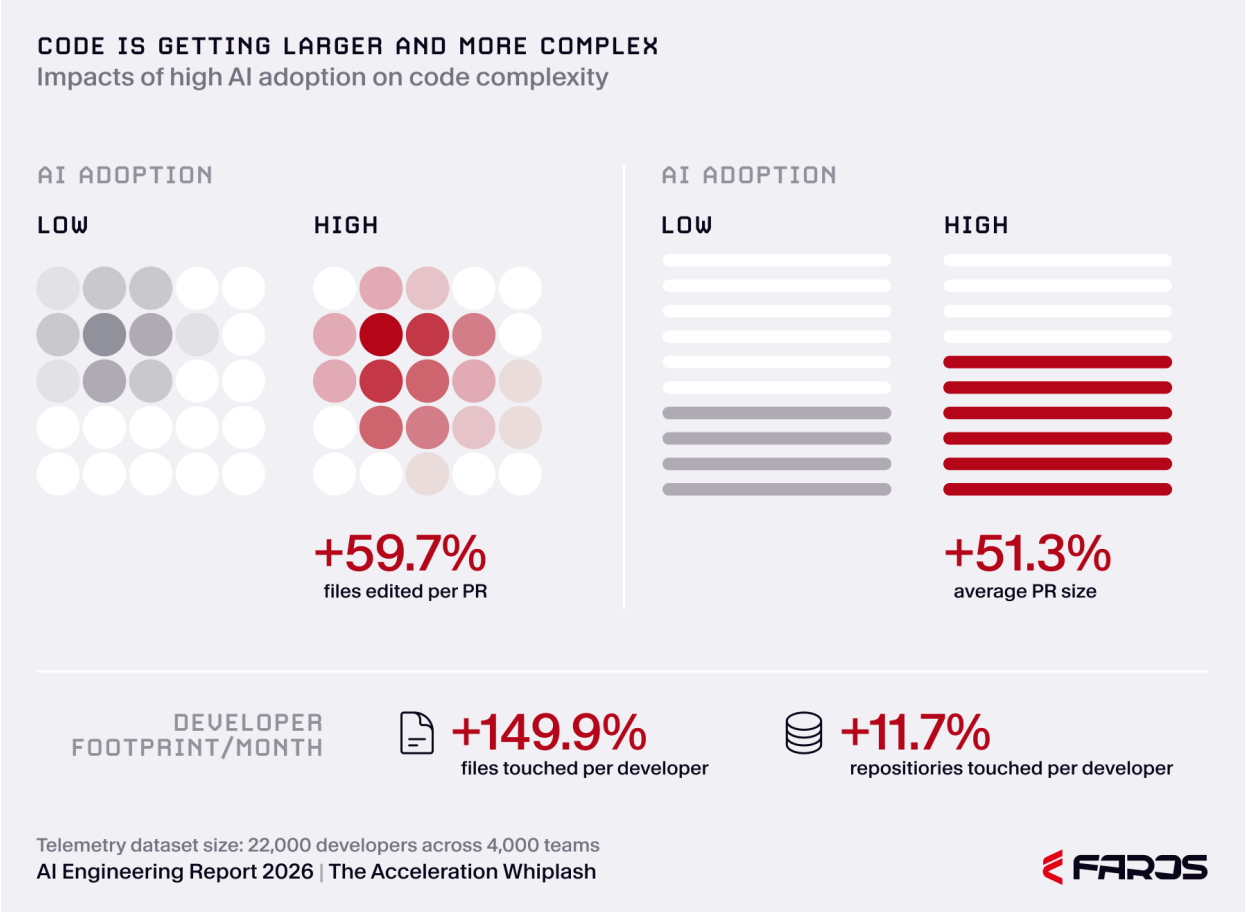
Code complexity: metric % change as teams move from low to high AI adoption

+51.3% average PR size

+59.7% average files edited per PR

+149.9% average files touched per developer per month

+11.7% average repositories touched per developer per month



PR size measures the number of lines of code added or removed in a single pull request. It is the most direct indicator of how large a change is being made to the codebase in one go. Small PRs have long been a best practice in software engineering. Smaller, more focused changes are easier to understand, safer to merge, and simpler to roll back if something goes wrong. A 51.3% increase in average PR size means larger, less predictable changes entering the codebase at once, with a wider blast radius if something breaks.[6]

The footprint of a typical code change has expanded significantly. The average number of files edited per PR is up 59.7%, meaning each change reaches further into the codebase than before. At the developer level, the monthly footprint has grown even more dramatically: files touched per developer per month are up 149.9%, and repositories touched per developer per month are up 11.7%. Each AI-assisted code change is larger, touches more of the codebase, and carries a wider blast radius than before.

> **Each AI-assisted code change is larger, touches more of the codebase, and carries a wider blast radius than before.**

Several factors are likely contributing to this expansion. Coding agents may be updating related files a human would have missed, or extending scope beyond what the task requires. AI is enabling long-deferred refactoring projects that touch broad surface areas by design. And developers, newly empowered to work in unfamiliar parts of the codebase, may lack the judgment to constrain what the AI touches.

To shed light on the root cause, organizations should utilize their engineering data to track lines of code per story point, PR size by task type, and rework rate by PR size over time. If larger PRs are taking disproportionately longer to review and producing more incidents, you have a case for enforcing size limits at the tooling level.

[6] PR size is up 51.3%, down from the 154% increase in our prior report. The two datasets differ in size, composition, and time period, so direct comparison should be treated as directional. What is consistent across both is the direction: PR size is up substantially when AI adoption is high, and the other complexity metrics in this dataset confirm the pattern.

FINDING #5: PRE-MERGE CODE QUALITY

Code quality before merge is declining

Reviewers are catching more. They are also being overwhelmed by poor code quality.

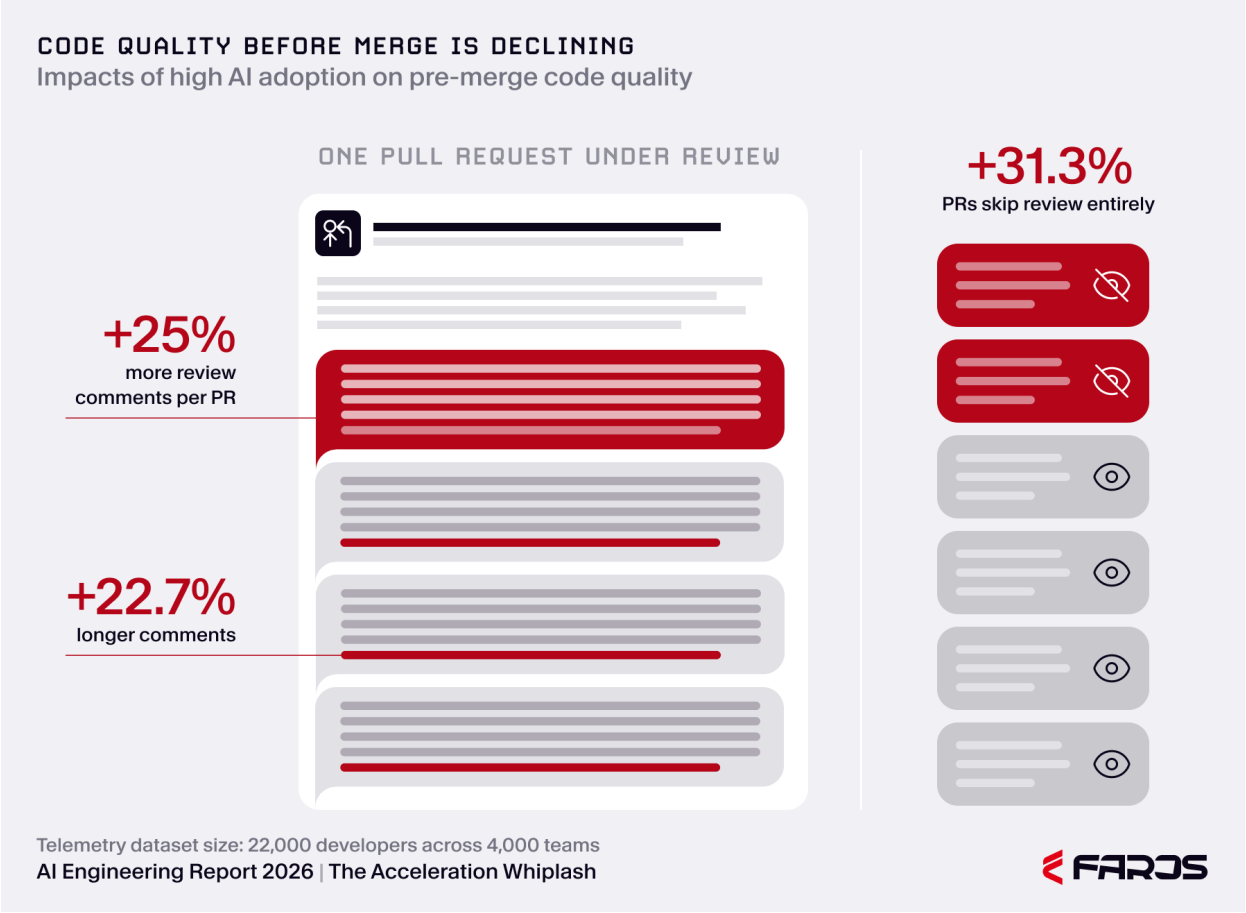
The signals around code quality at the point of review have deteriorated. Yet, dangerously, more PRs are being merged without any review—human or agentic.

Pre-merge code quality: metric % change as teams move from low to high AI adoption

+25% average review comments per PR

+22.7% average length of PR review comments

+31.3% PRs merged without any review



PR review comments per PR are up 25%, and average comment length is up 22.7%. Two caveats are worth stating before drawing conclusions. First, more code is being written per PR, so some increase in

comment volume is expected even if the defect rate per line were unchanged. Second, agentic code review has grown from zero to 25% of PRs in this dataset, and these comment statistics do not distinguish between comments left by humans and those left by agents. Agents are known to comment more verbosely and on a broader range of issues including structure, formatting, and style, which may be inflating both the volume and length of comments independently of underlying code quality.

With those caveats in mind, the most direct interpretation of rising comment volume and length is still that the code arriving for review is requiring more scrutiny than before. The next section examines review and workflow timing data, which is not subject to the same confounds and tells a consistent story: every stage of the workflow is taking significantly longer, and the bottleneck is most severe at the point of review.

The 31.3% increase in PRs merged without any review is the most urgent finding in this section. Code is entering production without oversight at a meaningfully higher rate as AI adoption has scaled.

The 31.3% increase in PRs merged without any review is the most urgent finding in this section. Code is entering production without oversight at a meaningfully higher rate as AI adoption has scaled.

FINDING #6: FLOW AND EFFICIENCY

The workflow is slowing down at every stage

More work entered the queue. The humans managing it could not keep up.

The metrics in this section, with the exception of Lead Time, come from work management systems like Jira, Azure DevOps, and similar tools that track how tasks move through a team's workflow. The average time a task spends in progress has increased 225.2%.

As work advances from in progress to review to testing to done, each handoff is a moment when human attention, judgment, and capacity determine what happens next. Across every one of those stages, the time spent is up substantially.

Flow and efficiency: metric % change as teams move from low to high AI adoption

+225.2% average time a task spends in progress

+81.8% average time a task is waiting

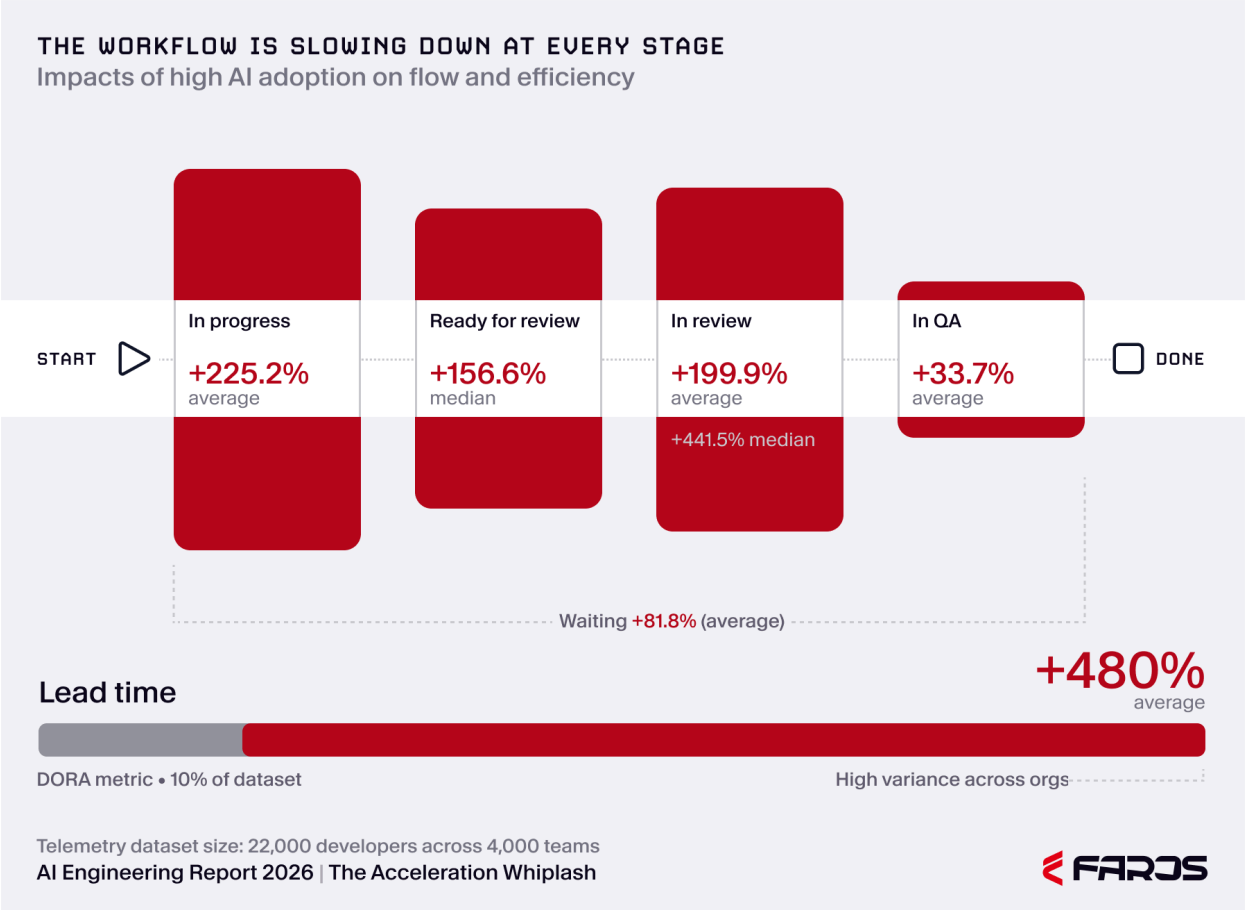
+156.6% median time to first PR review

+199.6% average time in PR review

+441.5 median time in PR review

+33.7% average time task is in QA

+480.4% Lead time from code commit to production (measured by 10% of dataset)



The 81.8% increase in average time a task is waiting reflects a specific category of workflow states—those explicitly classified as wait states, where work is queued and blocked rather than actively

being progressed. Different teams define their workflows differently, but across all of them, the time spent in states where nothing is moving has increased substantially. Work is not just taking longer to complete, it is spending more time in a state where no one is working on it at all.

> **Work is not just taking longer to complete. It is spending more time in a state where no one is working on it at all.**

Before a reviewer even picks up a pull request, the median wait time has grown 156.6%. Once someone does begin reviewing, the time spent in review has increased 199.6% on average and 441.5% at the median.

We've established that PRs are larger, touch more files, and contain more AI-generated code. But complexity alone does not explain the magnitude of this increase. In our view, the quality findings point to something more fundamental: the code arriving for review is often not review-ready. Reviewers are not just assessing more code, they appear to be working harder to bring that code up to a standard it should have met before the PR was opened.

The review time data also points to a specific and underappreciated consequence: the tax falling on senior engineers. AI-generated code presents a particular challenge for reviewers. It is often superficially convincing: idiomatic, well-named, stylistically consistent with the surrounding codebase. It looks like code written by someone who knows what they are doing. The structural and logical failures, when they exist, are beneath the surface. Catching them requires the reviewer to read carefully, reason about intent, and reconstruct the problem the code was meant to solve, rather than simply scanning for obvious errors. This is slow, expensive cognitive work. It is the work that senior engineers are uniquely equipped to do, and it is consuming them.

The review time data also points to a specific and underappreciated consequence: the tax falling on senior engineers. AI-generated code presents a particular challenge for reviewers. It is often superficially convincing: idiomatic, well-named, stylistically consistent with the surrounding codebase. It looks like code written by someone who knows what they are doing. The structural and logical failures, when they exist, are beneath the surface. Catching them requires the reviewer to read carefully, reason about intent, and reconstruct the problem the code was meant to solve, rather than simply scanning for obvious errors. This is slow, expensive cognitive work.

The result is a pyramid under pressure from below: more output than the top can review, arriving faster than the review process was designed to handle. The engineers with the deepest knowledge of the system are spending their most valuable hours unraveling plausible-sounding code that should never have reached them in the state it did.

Note: many organizations in this dataset use automated workflow nudges to surface stalled work. These findings reflect environments where teams are already intervening. Without that automation, the numbers are likely higher.

A different instrument, the same signal

Lead time measures the time from code commit to production deployment and is captured from version control and CI/CD pipeline data, not work management. Unlike a Jira "done" status, which is set by a human marking work complete, lead time reflects what actually happens in the delivery infrastructure: how long it takes for merged code to clear the build, test, and deployment pipeline and reach a live environment.

For the organizations that instrument this, lead time has increased 480.4%.[7] The variance across organizations is high, so the specific figure should be treated directionally. But the direction is unambiguous: the deployment pipeline is under the same pressure as every Jira stage above. Code is completing, but it is not reaching production any faster.

[7] Lead time is measured by approximately 10% of the dataset. High variance between teams is present. We include it as directionally consistent with the broader pattern, but recommend caution in applying this specific figure.

FINDING #7: CODE QUALITY IN PRODUCTION

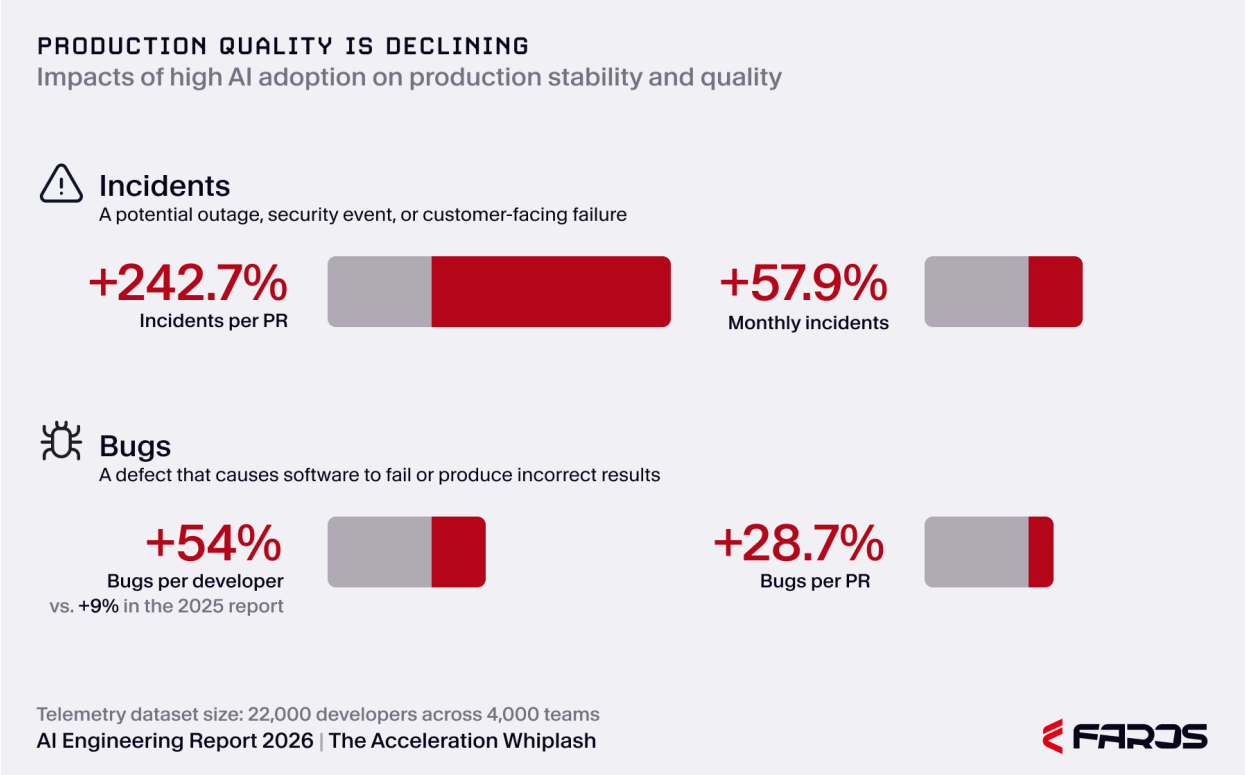
Poor quality code is reaching production

The bugs are real. The incidents are real. The code entering production systems is not meeting the bar that engineers once set for themselves.

AI-generated code is reaching production, and it is not holding up. Incidents have tripled relative to the low AI adoption baseline. This represents failures in systems that real users depend on.

Production code quality: metric % change as teams move from low to high AI adoption

- **+242.7%** incidents per PR
- **+57.9%** monthly incidents
- **+54%** bugs per developer, up from +9% in our prior report
- **+28.7%** bugs per PR
- **+12.6%** percentage of reopened Jira tickets



The incidents-to-PR ratio of +242.7% means that for every PR merged, the probability of an incident has more than tripled relative to the low AI adoption baseline. Every incident represents a potential outage, security event, or customer-facing failure. AI-generated code is now running in production systems across every industry, including finance, healthcare, and infrastructure and driving a 57.9% increase in monthly incidents.

The incidents-to-PR ratio of +242.7% means that for every PR merged, the probability of an incident has more than tripled relative to the low AI adoption baseline.

The bugs-per-developer figure warrants particular attention. In our prior report, this metric increased 9%. In this dataset, it has increased 54%. This is a concerning acceleration. More AI-generated code in the codebase correlates with more bugs per developer, and that relationship is strengthening as adoption deepens. Bugs-to-PR-ratio is also up, meaning more of the code being shipped is arriving with defects. A question we intend to examine in future editions is whether the increase in bugs and incidents persists when normalized for PR size, or whether larger PRs account for the majority of the quality deterioration.

Reopened Jira tickets are up 12.6%. A ticket is reopened when work that was marked complete is later found to be insufficient—typically caught in QA or after deployment. It is a strong quality signal precisely because the work cleared someone's definition of done before being sent back.

There has been industry debate about whether junior engineers are still needed, given AI's ability to handle entry-level coding work. The production quality data challenges something more uncomfortable than that assumption. The quality gap is not being caught at the point of review, and review is where senior engineers hold authority. Whether the cause is volume, tooling, or misplaced trust in AI output, the result is the same: declining quality, accelerating incidents, and more unreviewed merges reaching production. To be clear, the problem is not who is reviewing the code. It is that the code arriving for review was never ready. This is an authoring problem, not a review problem.

WHAT ENGINEERING ORGANIZATIONS SHOULD DO

AI is the primary author now. Here is what that requires.

The data in this report confirms two things simultaneously. AI coding tools are delivering real throughput gains: more tasks completed, more epics shipped, more code written than at any prior point in our dataset. And the code being written is not production-grade. More bugs are reaching production. Incidents have tripled per PR merged. The engineering systems built around human-paced development and human-quality code were not designed to absorb what AI is now producing.

Both of these things are true at once, and any response that addresses only one of them will fail.

Agentic authoring currently accounts for less than 1% of PRs in this dataset. That is, for now, a good thing. The data in this report reflects what happens when developers use AI as a primary authoring tool with humans still in the loop. Remove that human from the loop entirely, and every metric here faces pressure that is an order of magnitude greater. The industry is not ready for that transition. Neither are the tools.

The instinct to tighten review is the wrong response.

More reviewers, stricter gates, longer QA cycles treat the symptom. Review exists to catch mistakes. The goal should be fewer mistakes arriving at review, not more humans deployed to catch them. The answer is to raise the quality of code at the point of authoring. When AI tools operate with richer context, codebase standards, architectural intent, implementation guidelines, and testing requirements built in, the code they produce is better from the start.

You can see it. Now act on it.

The organizations represented in this data have something most of the industry does not: visibility into what is actually happening across their engineering environments. They can see where work stalls, where quality slips, and where incidents cluster. That is an advantage, and it is not a small one.

But visibility alone is not enough. The next step is control—not control in the sense of slowing AI down, but control in the sense of equipping AI to do its job better. An AI harness that guides agents with guardrails specific to the organization: codebase standards, architectural intent, security constraints, system history. That equips agents to test and validate their own output before a human ever sees it.

That orchestrates human collaboration when needed. And that creates feedback loops so that what gets accepted and what gets rejected makes the next output better.

That is the work ahead. The gap between knowing and acting is the only gap that matters now.

What comes next: How engineering leaders can respond

The data in this report is diagnostic. Below are ten recommendations to translate it into action. Not every metric applies to every organization at the same level of urgency. Start with the ones you can measure today.

1. Measure your incident-to-PR ratio and your monthly code change volume against monthly incidents.

Track both. The incident-to-PR ratio tells you how often a merged change results in a production failure. Monthly incidents normalized against lines changed tells you whether your overall risk exposure is rising as AI output scales. If either has more than doubled since you began scaling AI adoption, your review gates are not keeping pace. Pair both with bug severity data, so you can filter across all severity levels, not just critical incidents. This will provide a complete picture of production quality trends.

2. Define a review policy. Then enforce it as a gate.

A 31.3% increase in PRs merged without any review is not sustainable. But the answer is not simply "every PR must have a human review." As AI adoption scales, that requirement will break under the weight of volume. The decision engineering leaders need to make is more nuanced: which areas of the codebase require human review, which can be reviewed by a trusted AI agent, and which require both. Slice by risk: high-stakes systems, security-sensitive paths, and customer-facing services warrant human eyes. Lower-risk areas may be appropriate for agent review. What is not acceptable under any policy is no review at all. Every PR must pass a gate—human, agent, or both—before it merges. Make that gate explicit, enforce it at the tooling level, and audit it regularly.

3. Set PR size guidelines for your coding agents and enforce them.

Average PR size is up 51.3%. Large PRs compound the senior engineer tax: they are harder to review, carry a wider blast radius, and are more likely to contain structural errors that surface-level reviews will miss. Best practice for agentic development follows a research→ plan→ implement sequence. The plan phase must produce a scope that results in a PR that can be deployed to production as a distinct, self-contained unit of value. It must be able to pass the full pipeline independently. Build these

constraints into your agent instructions. If your coding agents are not operating within explicit PR size guidelines, they are generating scope beyond what your review process can safely absorb. Examine whether your incident and bug rates correlate with PR size in your environment. If larger PRs produce disproportionately more failures, you have a case for hard size limits before a PR can be opened.

4. Build quality gates into the development environment, not the review queue.

Do not let structurally complex or incomplete code reach your senior engineers' review queue. AI agents should be instructed to iterate not just against the completion criteria in the ticket, but against a defined set of quality gates: static analysis thresholds, linting rules, test coverage minimums, and the success criteria defined in tests. The goal is test-driven development at the agent level. The agent's definition of done is passing a set of tests that represent the specification of the task, not just producing working code.

5. Use workflow stage times as input to a continuous retrospective process.

Time spent in progress, in review, and in QA are universally tracked across work management systems. Monitoring them is not enough. Deploy agents that examine time spent at each stage on a regular cadence, identify patterns, surface root causes, and generate hypotheses about what is preventable.

For example, when a task spends an unusually long time in review, an agent should ask why. Was the PR too large? Was the context insufficient for the reviewer to make a decision quickly? Was the work presented in a way that required the reviewer to dive into raw code and logs rather than reading a clear summary? In many cases, the friction is not the work itself. It is how the work is being presented. Agents can surface what matters at each stage in a more digestible form, reducing the cognitive load on the engineers making decisions.

This is a continuous improvement loop, not a one-time audit. The agent examines the signal, proposes an optimization, and the next cycle tests whether it worked. Time spent in each workflow stage becomes an input to an always-on retrospective process that does not require a sprint ceremony or a manually compiled report. The goal is a pipeline that gets measurably faster over time, driven by agents that learn from every cycle where the friction was and what could be done differently.

6. Watch work restarts as a signal about agent context quality.

A work restart occurs when a task returns to in-progress after moving to another stage. Rising restarts under high AI adoption are a signal that coding agents took the wrong path and work had to start over, which points directly to insufficient context at the authoring stage. If your restart rate is climbing

alongside your AI adoption, examine what context your agents are operating with. The fix is upstream: better context provisioning before the agent begins, not manual correction after the restart.

7. Distinguish between human and agent review comments and act on both differently.

More comments per PR and longer comments signal that reviewers are working harder. But not all comments carry equal weight. Agent review comments tend to address structure, formatting, and surface-level code patterns which are valuable, but best caught earlier in the process, before the PR is opened. Human review comments represent expensive senior engineer intervention and are the signal that matters most. When a human reviewer identifies something that should change, that finding should feed back into the agent's harness and be turned into a generalized rule that prevents the same issue in future PRs. Every human comment is an opportunity to improve the authoring stage. Treat it that way.

8. Build an automated investigation loop around code churn.

An increase in the ratio of lines deleted to lines added for recently merged code is a signal worth taking seriously and one that warrants automated investigation, not just monitoring. Every time significant churn occurs, the system should ask: what are the leading causes, which areas of the codebase are affected, and is this churn preventable or acceptable? Build a hypothesis, examine the pattern, and feed the findings back into agent instructions. Code churn that is preventable is exactly the kind of signal that should improve your authoring process over time, automatically and continuously.

9. Repurpose engineering capacity. Do not reduce it.

Do not rush to cut headcount on the basis of AI output gains. The risk is that you reduce staff, quality problems compound, and you are forced to rehire within months. The engineers you are considering cutting are in many cases the ones absorbing the quality gap AI is creating. The right move is repurposing, not replacing: redirect engineering capacity toward building the scaffolding that makes AI adoption sustainable. Develop review systems, quality infrastructure, context provisioning, and the automated retrospective loops that make the system smarter over time.

The amount of code entering production is higher than ever. So is the risk to the business from quality, security, and compliance failures. Stabilize the process before making workforce decisions. Leverage these signals to demonstrate that the process is under control, and then make decisions from a position of confidence, not urgency.

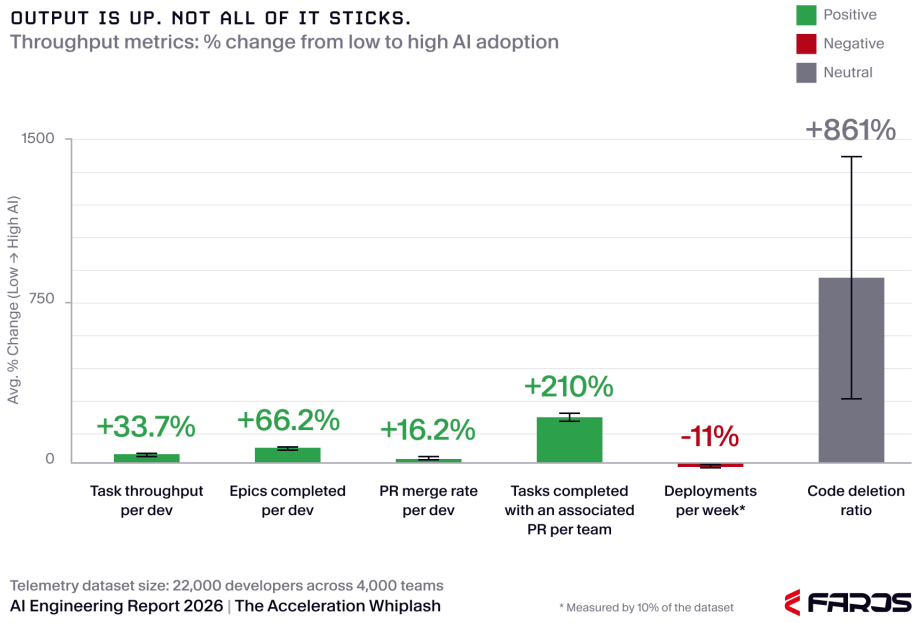
10. Build a context engine for your engineering environment.

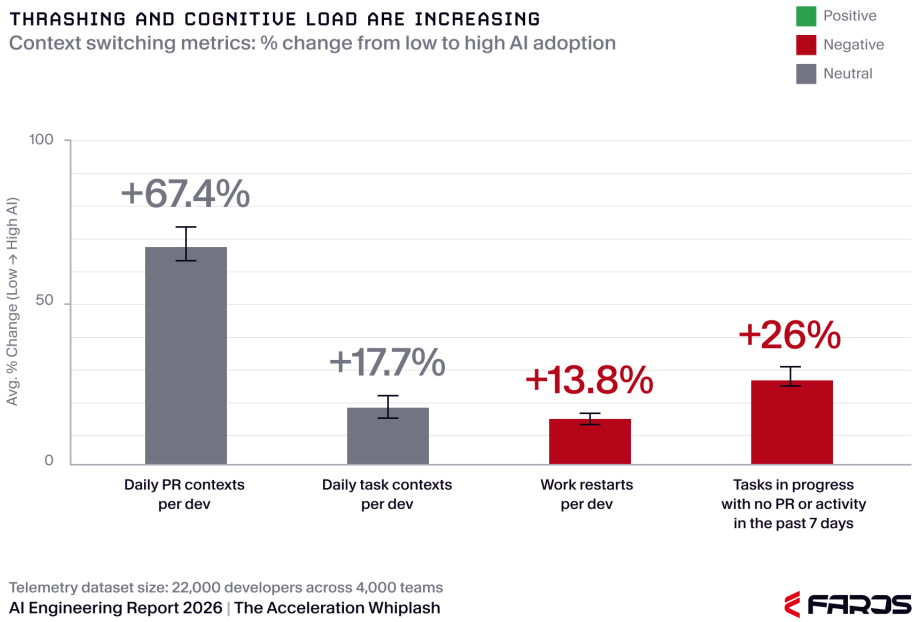
The fix is upstream. AI agents cannot derive intent by looking at the current state of a codebase. The codebase is just a point in time. Intent is derived from how the codebase evolved: the history of changes, the reasoning behind each one, the architectural decisions that accumulated over years of development. That historical context, combined with codebase standards, security constraints, and testing requirements, is what AI agents are currently missing. Building it into the authoring environment creates the conditions where AI can produce code that requires less human intervention to reach a shippable state. Without it, the patterns documented in this report will continue, and will worsen as agentic authoring scales.

These recommendations assume a baseline of visibility into your engineering environment. If you cannot answer most of these questions today, the visibility gap is the first problem to solve. You cannot govern what you cannot see.

APPENDIX

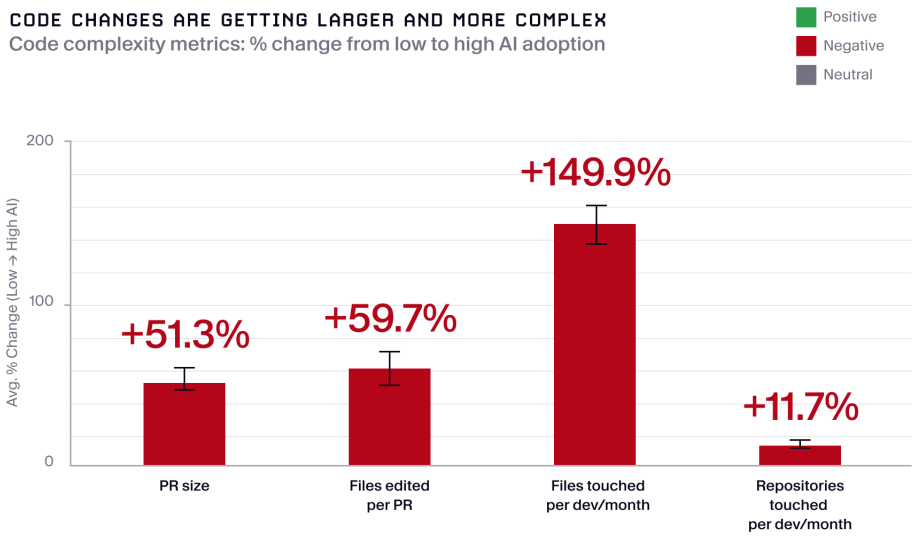
Supporting Charts





CODE CHANGES ARE GETTING LARGER AND MORE COMPLEX

Code complexity metrics: % change from low to high AI adoption

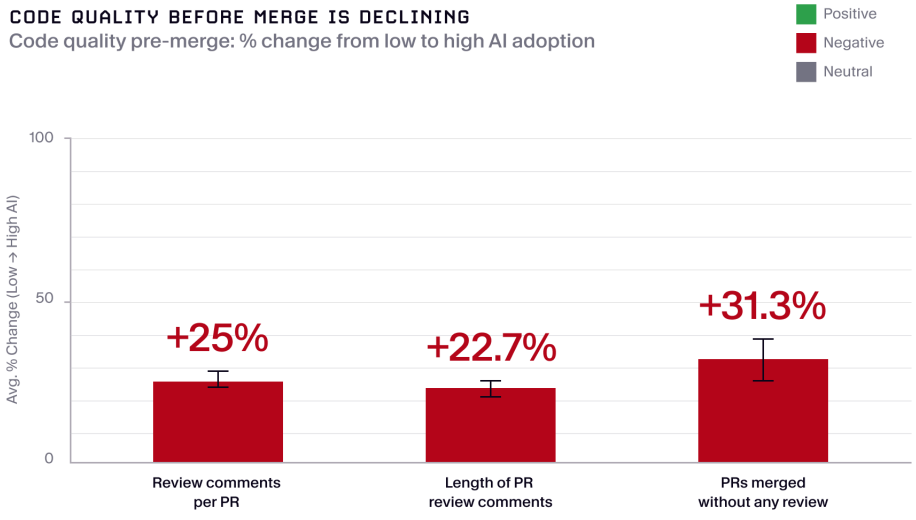


Telemetry dataset size: 22,000 developers across 4,000 teams

AI Engineering Report 2026 | The Acceleration Whiplash

CODE QUALITY BEFORE MERGE IS DECLINING

Code quality pre-merge: % change from low to high AI adoption

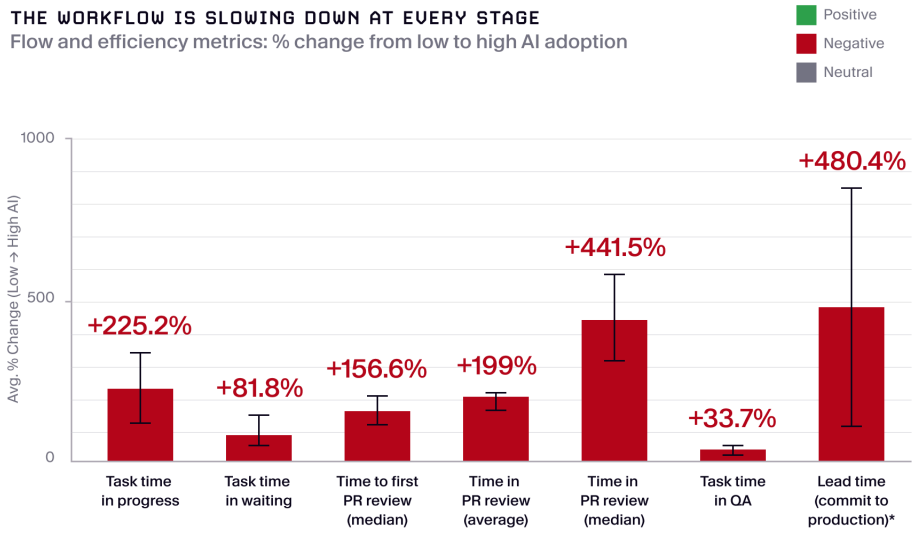


Telemetry dataset size: 22,000 developers across 4,000 teams

AI Engineering Report 2026 | The Acceleration Whiplash

THE WORKFLOW IS SLOWING DOWN AT EVERY STAGE

Flow and efficiency metrics: % change from low to high AI adoption



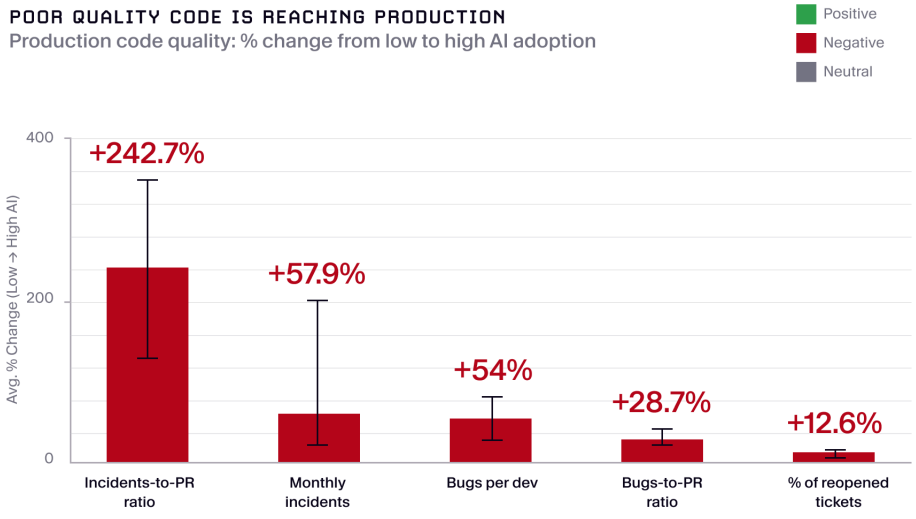
Telemetry dataset size: 22,000 developers across 4,000 teams

AI Engineering Report 2026 | The Acceleration Whiplash

* Measured by 10% of the dataset

POOR QUALITY CODE IS REACHING PRODUCTION

Production code quality: % change from low to high AI adoption



Telemetry dataset size: 22,000 developers across 4,000 teams

AI Engineering Report 2026 | The Acceleration Whiplash

About Faros

Faros is the system for running engineering with AI. We give engineering leaders visibility into how work operates across code, people, and systems, and control over how that work progresses through enforceable workflows and policy. This enables organizations to deploy AI effectively and improve engineering throughput with stronger cost efficiency.